JS FUNDAMENTALS IN ADVANCE TOPIC:

Hacker Hero Challenges

If you haven't completed Hacker Hero's Learn to Code and Basic Algorithm courses, go ahead and do that now. If you've completed that course but didn't do the challenges in Javascript, go ahead and re-do all the challenges, this time in Javascript.

If you have done these challenges in Javascript before, but still weren't too comfortable with solving these challenges, go ahead and do the challenges again. You could create a new account and re-do the challenges, making sure that you can complete each challenge all under a few minutes.

Once you've finished all the challenges for Learn to Code and Basic Algorithm, then go ahead and start working on Hacker Hero's Adv. Algorithms and Data Structures and finish all challenges up to all the Sorting Challenges.

In other words,

- Complete Hacker Hero's Learn to Code Course in Javascript - make sure you can do all challenge under a few minutes each
- Complete Hacker Hero's Basic Algorithm Course in Javascript - make sure you can do all challenges under 5 minutes each
- Complete Hacker Hero's Adv. Algorithm Course in Javascript for Recursion, Dynamic Programming, and Sorting. Do NOT do any challenges yet under Nodes, Singly Linked List, Queues and Stacks, Doubly Linked List, Binary Search Tree, Hash Table, or Design Patterns yet

ES5

EcmaScript (ES) is a standardized scripting language for Javascript (JS). Most browsers released after 2012/2013 support ES5. ES6, also called ECMAScript 2015, is a newer version that are supported by major browser released after 2016-2017. ES6 introduces some significant changes to how developers code in Javascript.

For you, you need to understand both ES5 and ES6. A lot of code you'll see will be built in ES5, especially if the code-base was initially built a few years ago.

For this course, we'll focus initially on ES5 syntaxes and introduce ES6 syntaxes in a bit, after Object Oriented Programming concepts are introduced.

ES5

The courses at Hacker Hero were mostly done using ES5 syntaxes. Although ES6 has a lot of additional features, for now, we'll have you focus on just a few ES6 features for now.

Remember that in Javascript, the core building blocks are done with the following syntaxes:

1. Variables

```
var a = 5;   // for storing a number in a variable
var b = "hello world";  //for storing a string in a variable
var c = [1,3,5,7,8];  //for storing an array in a variable
```

2. If/Else Statements

```
if( ...condition... ) {
  // codes that should be executed if the condition is true
}
else if( ...condition... ) {
  // codes that should be executed if the second condition is true
}
else {
  ...
}
```

3. For loops

```
for(var i=0; i<10; i++){
    console.log(i);  //prints a number from 0 to 9
}
```

4. Functions

```
function abc(x, y, z){
    ...
    return x+y+z;
}
```

Objects

A variable type that we haven't used very much yet is a type called Objects. In other programming languages, this is often called as an associative array, hash, or a dictionary. In Javascript, it's simply called as Objects.

An object has a key value pair. For example:

```
var teacher = { name: 'Michael', city: 'Seattle', state: 'Washington' };
```

We can access each value by doing:

```
console.log( teacher.name );
console.log( teacher.city );
```

We can update the value by doing:

```
teacher.city = 'Bellevue';
console.log( teacher.city ); //this will now log Bellevue
```

You could even create a new key/value by adding a new property. For example:

```
teacher.favorite_sport = "soccer";
console.log(teacher); //this would now log all the key values storied in the variable teacher
```

An Array of Objects

It's also very common to have a variable be an array of multiple objects. For example,

```
var students = [ { name: "John", age: 25 },
{ name: "KB", age: 21 },
{ name: "Jomar", age: 25 } ];
```

If you wanted to loop through this array and print each value in the object, you could do something like this:

```
for(var i=0; i<students.length; i++){
    console.log( "Student Name: " + students[i].name + " - Age: " + students[i].age);
}
```

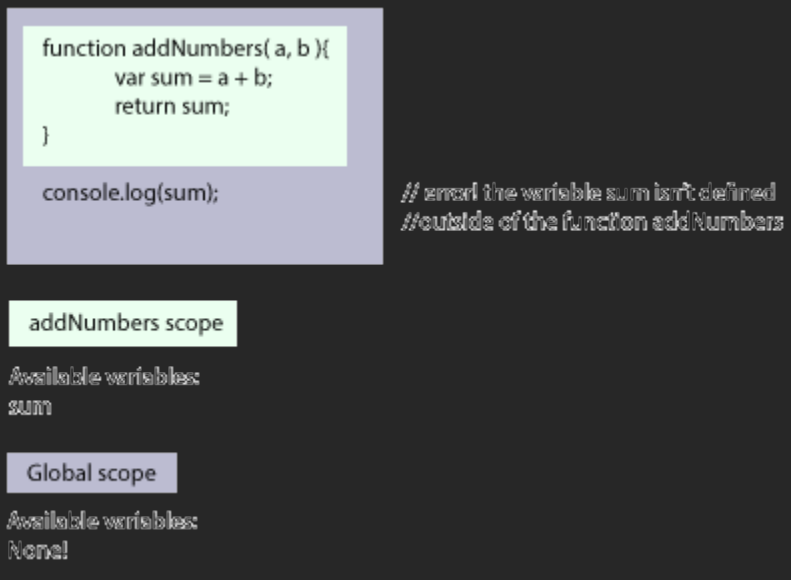
You could alternatively, do something like this also:

```
for(student of students){
    console.log( "Student Name: " + student.name + " - Age: " + student.age);
}
```

Scope and Hoisting

Scope

The understanding of scope is a key part to your growth as a JS developer. Read this section slowly, and refer back to it often, as some of the most common misunderstandings of JS stem from scope problems. At a base level, JavaScript has **function scoping.** This means that when we declare variables inside of a function, they are only accessible to that function. *Function calls create new scope.* Consider the below diagram:



In this example, we have two levels of our scope. *Global scope* refers to everything in our global namespace. In this example, Global contains no variables. The *addNumbers scope* refers to all the variables that exist within that function call.

Note: *Global* scope is never actually empty, however. Anytime we declare a variable globally, it needs to share namespace with all of the objects JavaScript already makes available to us, like `console`, `document`, and `Math`.

Now let's see something more complex:

```
function addNumbers( a, b ){
        var sum = a + b;
        return sum;
}

function addArrayElements( array ){
        var array_sum = 0;
        var array_length = array.length;
        for (var i = 0; i < array_length; i++) {
                addNumbers( array_sum, array[ i ] );
        }
        return array_sum;
}

var new_sum = addArrayElements ( [3, 4,5] );
```

addNumbers scope

Available variables:
sum

addArrayElements scope

Available variables:
array_sum
array_length
}

Global scope

Available variables:
new_sum

Finally, consider what would happen when we start nesting functions:

```
function addArrayElements( array ){
        var array_sum = 0;
        var array_length = array.length;

        for (var i = 0; i < array_length; i++) {
                addNumbers( array_sum, array[ i ] );
        }

        function addNumbers( a, b ){
                var sum = a + b;
                return sum;
        }

        return array_sum;
}

var new_sum = addArrayElements ( [3, 4,5] );
```

addNumbers scope

Available variables:
sum
array_sum
array_length
i

addArrayElements scope

Available variables:
array_sum
array_length
i

Global scope

Available variables:
new_sum

Key Takeaways

1. Each function has access to all the variables in its parent function.
2. No function has access to the variables in its child functions.
3. Your entire .js file can be thought of as the outermost function or 'global' scope.
4. With ES6, we can also take advantage of block-scoping.

Hoisting

Imagine the following code:

```
console.log( whoamI );
```

What would be the result? Go ahead and run this, and you'll see that the Javascript interpreter will respond "***Uncaught ReferenceError: whoamI is not defined***". This makes sense as that variable whoamI was never defined.

Now, what would happen if you did the following?

```
console.log( whoamI );
var whoamI = 5;
```

Would it still say "***Uncaught ReferenceError: whoamI is not defined***"? After all, whoamI variable was never defined until the second line right?

The answer is that it does NOT. Instead, it says simply that whoamI is 'undefined' (meaning that variable actually exists and Javascript recognized that variable even though it was never defined until the following line)! What actually happens is what's called Javascript Hoisting. Whenever a variable is created using 'var', it's as if that variable was created at the top of its scope. For example, Javascript would really interpret the top code as follows:

```
var whoamI;
console.log ( whoamI );
whoamI = 5;
```

Note that the variable declaration 'ballooned to the top'.

Understanding Scope and Hosting

Without using a computer, but using plain paper and T-diagram, predict the output of the following code:

Example 1

```
var a = 10;
function abc() {
    var a = 15;
    console.log('a is', a);
}
console.log('a really is', a);
```

Example 2

```
var a = 10;
function abc() {
    var a = 15;
    console.log('a is', a);
}
abc();
console.log('a really is', a);
```

Example 3

```
if(a == undefined) {
   console.log("a is declared but hasn't been set to a specific value yet");
}
```

Example 4

```
if(a == undefined) {
   console.log("a is declared but hasn't been set to a specific value yet");
}
var a = 15;
```

Example 5

```
var a = 15;
function abc(a){
   return a+10;
}
var final = abc(a);
console.log('final is', final);
```

Example 6

```
var a = 15;
function abc(){
   a = a+10;
}
console.log('a is', a);
```

Example 7

```
var a = 15;
function abc(){
   a = a+10;
}
console.log('a is', a);
abc();
console.log('a is', a);
```

Example 8

```
var a = 15;
function abc(){
   var a = a+10;
}
console.log('a is', a);
abc();
console.log('a is', a);
```

Example 9

```
var a = 15;
function abc(a){
   a = a + 15;
}
console.log('a is', a);
abc();
console.log('a is', a);
```

Once you've written what you think the output for each of these examples are, run each example on JSBin or save these as a javascript file and run each file using Node.js.

Any surprises on what the computer actually logged vs what you predicted? If you didn't get the right answers on all of the examples above, re-read Scope and Hoisting section to learn more and try again.

One key thing to remember is that 1) whenever a function is called, it creates a new T-diagram for that function, and 2) any variable that was declared inside that function, as soon as the interpreter leaves the function, that T-diagram for the function is thrown out!

ES6 Offers Block Scoping

ES6 gives us the ability to create variable with `const` and `let` in addition to `var`. While `var` is function scoped, `let` and `const` are also **block scoped.** For example, if a `let` or `const` variable is defined inside an if-block or a

loop, then it will not be defined outside of that block. Referring to that variable outside of the block will produce a ReferenceError since it is not defined.

```
function letLoop(){
    for (let i=0; i<3; i++) {     // notice we use let to define i
        console.log(i);           // this will log 0, 1, 2
    }
    console.log(i);               //  this will give us a ReferenceError because i is not defin
ed outside the loop
}
```

In contrast, when we use var to define our iterator in a loop, then we may refer to it outside the loop.

```
function varLoop(){
    for (var i=0; i<3; i++) {     // notice we use var to define i
        console.log(i);           // this will log 0, 1, 2
    }
    console.log(i);               //  this will log 3
}
```

For variables where the value is not going to be changed, you can declare using const. If you try to update or change a value that was storied in a variable that was declared using 'const', it will throw an error.

For example,

```
const a = 15;
a = a + 10;  // this will throw a TypeError: Assignment to constant variable<div id="copy-tool
bar-container" style="cursor: pointer; position: absolute; top: 5px; right: 5px; padding: 0px
3px; background: rgba(224, 224, 224, 0.2); box-shadow: rgba(0, 0, 0, 0.2) 0px 2px 0px 0px; col
or: rgb(187, 187, 187); border-radius: 0.5em; font-size: 0.8em;"><span id="copy-toolbar">copy<
/span></div>
```

Most developers who are familiar with ES6 will typically use let and const and stop using var. Some of the older code base may still declare variables all using 'var'. For you, you should know how to use both and for us, going forward, we'll lean more heavily on using let and const instead of var.

Key Takeaways

1. Each function has access to all the variables in its parent function.
2. No function has access to the variables in its child functions.
3. Your entire .js file can be thought of as the outermost function or 'global' scope.
4. With ES6, we can also take advantage of block-scoping.

CallBacks

Callback is a critical feature of Javascript and what makes Javascript quite different from other traditional languages such as C, C++, C#, Java, Python, etc.

For example, consider the following code:

```
let a = 15;
let b = "hello";
let c = function(name) {
  console.log('my name is', name);
  return name.length;
}
```

All programming languages will allow you to store a number, a string, an array, or a hash/dictionary/objects in a variable, but would all languages allow you to store a function in a variable??? If you've come from a more traditional background (such as Java, C, etc), you will be puzzled why this is even allowed.

Not only is this allowed in Javascript (storing a function in a variable), this is what makes Javascript a powerful language to handle event-driven processes. Before, we get to that, however, let's get into the callback a bit more.

Level 1: Storing a function in a variable

To better understand callbacks, please use paper to predict the output for the following code. Do not run these code yet until you've predicted the output for all of the examples below.

Example 1

```
let c = function(name) {
  console.log('my name is', name);
  return name.length;
}
```

Example 2

```
let c = function(name) {
  console.log('my name is', name);
  return name.length;
}
c(15);
```

Example 3

```
let c = function(name = "Michael") {
  console.log('my name is', name);
  return name.length;
}
console.log( c() );
```

Example 4

```
let c = function(name = "Michael") {
  console.log('my name is', name);
  return name.length;
}
console.log( c("John") )
```

Now, run these code directly using Node.js and compare what was logged with what you predicted. Any differences? If so, really try to understand what's happening before you proceed next.

Level 2: Passing a function to a function

Okay, so storing a function as a variable, now you know that's perfectly acceptable in Javascript. Now, let's think about functions. Remember that for a function, I can pass a number, a string, or an array as an argument to the function? In other words, whatever you're used to storing in a variable, you also passed these as arguments to a function.

Now, if a function can receive a string, a number, or an array as an argument to a function, do you think it can also accept a function as an argument to its function???

For example, consider the code below:

```
const sum = function(a, b) {
    return a+b;
}
function performOperation(num1, num2, operation){
    let result = operation(num1, num2);
    console.log('result is', result);
    return result;
}
performOperation(3, 5, sum);
```

Go ahead and run this using paper and see if you can correctly predict the output. Isn't this interesting?

You could have written the code above to be simpler and make it like this:

```
function performOperation(num1, num2, operation){
    let result = operation(num1, num2);
    console.log('result is', result);
    return result;
}
performOperation(3, 5, function(a,b) {
    return a+b;
});
```

Isn't that really cool?

Level 3: Having a function return a function

Let's deepen how a callback function can be used even further. Remember that a function could return a number, a string, an array, an object, or anything that can be stored in a variable, right?

Do you think a function could also return a function? For example, what if someone wrote this and had JS interpreter run the code? What do you think the output would be?

```
function abc(a, b){
    return function() {
        console.log('hello');
        return a+b+10;
    }
}
let result = abc(5, 10);
console.log(result);
```

What would be the output for the following?

```
function abc(a, b){
    return function() {
        console.log('hello');
        return a+b+10;
    }
}
let result = abc(5, 10);
```

```
let z = result();
console.log(z);
```

Summary

A simple fact that we can store a function in a variable allows us to do some interesting things including passing a function to a function as well as having a function return another function!

This simple feature allows Javascript to handle a lot of event driven processes very well. This is also what allows Javascript to handle asynchronous events very well.

This behavior of having a function accept another function or having a function return a function is called 'callbacks'. Whether it executes that function that was passed/returned is up to the developer. In other words, just because a function was passed to a function or was returned from a function, it doesn't mean that the function would be executed. Most 'callback' functions are used in level 2 settings where a callback function is passed to a function. Level 3 callback, where a function returns a function is not used as frequently, although in certain situations, that could come in very handy.

As you get more experience with Javascript, you'll get better with callbacks and how to use them.

Callback Challenges I

Foreach

Create a function called foreach where the following code would work the way described below:

```
let result = foreach([1,2,3,4,5], function(num) { return num*2; });
console.log(result); //this should log [2,4,6,8,10]
//
result = foreach([1,2,3,"v88", "training"], function(val) {
    if(typeof(value) === 'number') {
        return 0;
    }
    else {
        return val;
    }
});
console.log(result); //this should log [0,0,0,"v88","training"]
//
result = foreach([1,2,3,"hello"], function(val) { return typeof(value); });
console.log(result); //this should log ["number", "number", "number", "string"]
```

Note how for the second foreach callback function, for a simple if/else statement, we can also use a ternary operator: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

If a ternary operator is used, above code could be simplified as follows:

```
let result = foreach([1,2,3,4,5], function(num) { return num*2; });
console.log(result); //this should log [2,4,6,8,10]
//
```

```
result = foreach([1,2,3,"v88", "training"], function(val) {
    return ( (typeof(value) === 'number') ? 0 : val);
});
console.log(result); //this should log [0,0,0,"v88","training"]
//
result = foreach([1,2,3,"hello"], function(val) { return typeof(value); });
console.log(result); //this should log ["number", "number", "number", "string"]
```

Filter

Create a function called filter where it filters out any value in the array that doesn't meet the condition as specified in the callback function. For example,

```
let result = filter([1,2,3,4,15], function(val) { return val<10; }); //this filters each value in the array and only allows values that is less than 10
console.log(result); //this should log [1,2,3,4]
//
let result = filter([1,2,3,4,15], function(val) { return val<3; }); //only allows values that is less than 3
console.log(result); //this should log [1,2]
```

Reject

Create a function called reject that acts the opposite of the filter function. For example, have it reject any value in the array that meets the requirement specified in the callback function. For example,

```
let result = reject([1,2,3,4,15], function(val) { return val<10; }); //rejects any value that is less than 10
console.log(result); //this should log [15]
let result = reject([1,2,3,4,15], function(val) { return val<3; }); //rejects any value that is less than 3
console.log(result); //this should log [3,4,15]
```

Key Takeaways

A lot of new Javascript developers use callbacks or functions that use callbacks without really understanding how to design these functions that accept callbacks. This is dangerous as it makes the developer too reliant on built-in functions and other people's codes without really understanding the fundamentals.

All built-in functions are codes that are built using core-building blocks of software and now, if you really understand how to use a callback function, you too can create all these functions yourself from scratch! This will make you a better developer and build confidence in your abilities. Never use any built in functions without thinking first if you could create that function from scratch yourself using just the core building blocks of software.

Objects and Callbacks

Note that generally, a callback is simply a function that is passed as an argument for the function (the level 2 callback as we discussed previously).

Now, remember that for a Javascript object, we could set up a key value pair where the value could be a string, a number, an array, or even another Javascript object. Now that you know what a callback function is, note that you can also attach a callback function to a key. For example,

```
var obj = {
  name: 'Michael',
  age: 39,
  hello: function() {
     console.log('Michael says hello!');
  }
};
obj.hello(); // this executes the function inside the hello property and logs 'Michael says he
llo!'
```

A callback function could reference its own property by using a special keyword called 'this'. For example,

```
obj.celebrate_birthday = function() {
   this.age++;
   console.log(this.name + " is now " + this.age + " years old");
}
```

'this' in the object references the object itself. More about 'this' will be introduced later when we go more in depth into JS Object Oriented Programming.

You could even pass another argument to a callback function. For example,

```
var obj = {
  name: 'Michael',
  age: 39,
  location: 'Seattle Washington',
  hello: function() {
     console.log('Michael says hello!');
  },
  update_location: function(new_location){
    this.location = new_location;
    console.log('location has been updated to', new_location);
  }
};
obj.update_location('San Fernando Philippines');
console.log('obj is now', obj); //see what gets logged
```

You will have more experience playing with Javascript object and later when you learn ES5 and ES6 syntaxes for Object Oriented Programming, you'll learn a lot of other various ways of creating a Javascript object. For now, just focus on what I've taught you here.

Ninja Battle

Let's have you work on some exercises to improve your understanding of JS objects and attaching simple callbacks.

Note that I will provide you some sample codes to start out with.

```
var ninja1 = {
  hp: 100,
```

```
  strength: 15,
  attack: function() {
     //your code
  }
}
var ninja2 = {
  hp: 150,
  strength: 10,
  attack: function() {
    //your code
  }
}
```

What I want you to do is to come up with a program that gets each ninja to take a turn, and to attack each other, for a total of 10 rounds. When a ninja attacks, it should return a random number between 0 to the strength of that ninja. For example, for ninja1, whenever attack method is invoked for ninja1, it should return a random number between 0 to 15. For ninja2, the attack method should return a random number between 0 to 10.

After each attack, your program should tell who attacked who and what the hp of each ninja is now. After all 10 rounds of attack, announce who the winner is (based on who still has the higher hp).

For example, once your program runs, it may display an output such as this:

```
===Round 1===
Ninja1 attacks Ninja2 and does a damage of 9!   Ninja1 health: 100.  Ninja2 health: 142
Ninja2 attacks Ninja1 and does a damage of 8!   Ninja1 health: 92.   Ninja2 health: 142
===Round 2===
Ninja1 attacks Ninja2 and does a damage of 3!   Ninja1 health: 92.  Ninja2 health: 139
Ninja2 attacks Ninja1 and does a damage of 10!  Ninja1 health: 82.  Ninja2 health: 139
...
...
...
===Round 10===
Ninja1 attacks Ninja2 and does a damage of 13!  Ninja1 health: 35.  Ninja2 health: 48
Ninja2 attacks Ninja1 and does a damage of 10!  Ninja1 health: 25.  Ninja2 health: 48
Ninja2 WINS!!!!!
```

Note that there are many ways to do this assignment. Try to keep it as simple as possible and avoid using any built-in function.

Note also that later when you learn more about OOP and about ES5/ES6 syntaxes, you'll see better ways to create objects than what you did above. However, don't focus so much about these other syntaxes yet as you will learn about these soon also.

For now, focus on the basic concepts that were presented to you and figure out ways to complete this assignment.

Event Listeners

Now that you have a bit more experience with callbacks, let's have you learn a bit more about event listeners. Note that Javascript was built initially to handle events for web applications. It can, now with Node.js, do a lot more than that, but for now, let's focus on this nature of Javascript and how it can handle event.

Note that when you were using jQuery, you added event listeners to a specific HTML element. For example, if you have a button, and when it is clicked, you could use jQuery to perform certain actions.

In Javascript, attaching an event listener is quite simple. For example:

HTML

```
<button id='myBtn'>My Button</button>
<p id='demo'>...</p>
```

Javascript

```
document.getElementById("myBtn").addEventListener("click", function() {
    document.getElementById("demo").innerHTML = "Hello World";
});
```

There are lots of different events that the browser will listen. Once there is an event, the browser will then check if the element has a callback function attached for that event. If yes, it will execute the callback function.

Main Events

There are lots of HTML DOM events that the browsers listens to. Again, if certain elements have event listeners and a callback function attached to that event, the browser will execute the callback function attached to that event listener.

The mouse common HTML events are:

- change - AN HTML element has been changed
- click - the user clicks an HTML element
- mouseover - the user moves the mouse over an HTML element
- mouseout - the user moves the mouse out/away from an HTML element
- keydown - the user presses a keyboard key
- load - the browser finishes loading the page
- submit - a form is submitted

There are dozens of other HTML events that the browser will listen to. Take a quick look at W3School's list of all events: https://www.w3schools.com/jsref/dom_obj_event.asp (but spend no more than 10 minutes browsing through this list).

Using Inspect Element

Using Inspect Element, you can also see which html elements have event listeners attached. For example, see https://umaar.com/dev-tips/24-view-event-listeners/. This could come in handy when you're debugging your applicaiton.

querySelector & querySelectorAll

Javascript introduced some really neat functions to grab specific HTML elements.

QuerySelector

This is used to grab the first specific element specified and the general syntax is as follows:

```
document.querySelector(CSS selectors);
```

For example, look at how this is used for different CSS selectors.

```
document.querySelector("h1"); //gets the first <h1> element in the document
document.querySelector("p.example"); //gets the first <p> element in the document with class '
example'
document.querySelector("#title").innerHTML = "New Title"; //gets the element with an id of tit
le and updates the HTML
```

QuerySelectorAll

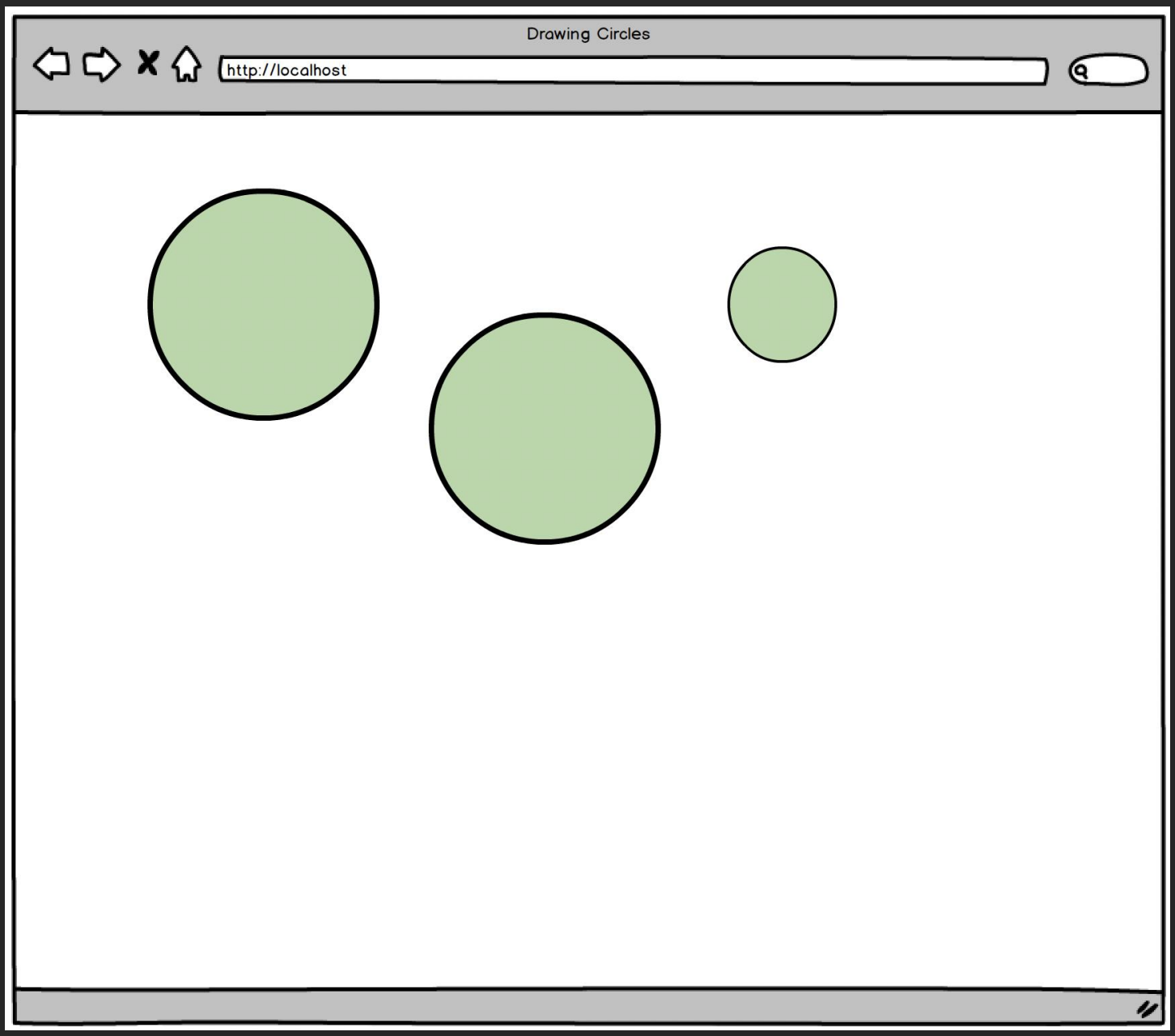
You can grab ALL elements specified by the CSS selectors using QuerySelectorAll. Note that QuerySelector only returns the first specific element. If you wants to grab ALL elements that meet the CSS selectors criteria, use QuerySelectorAll. For example,

```
let x = document.querySelectorAll("p.red"); // gets all <p> elements in the document with clas
s red
for(let i=0; i<x.length; i++){
   x[i].innerHTML = "hello"; //updates each innerHTML
}
//
let y = document.querySelectorAll("h2, p, img"); //gets all the <h2>, <p>, and <img> elements
for(let i=0; i<x.length; i++){
   x[i].style.backgroundColor = "blue"; //updates each style.backgroundColor to be blue
}
//
let z = document.querySelectorAll("div > p"); //gets every <p> element where the parent is a <
div> element
for(let i=0; i<x.length; i++){
   x[i].style.backgroundColor = "green";
}
```

Drawing Circles

Now that you know how to create eventListeners directly using Javascript, create a web application where it initially shows a blank white screen. Then, whenever a mouse is clicked, have a circle of random radius between 10px - 200px

appear on the screen where you clicked. Have the default color of the circle be green. For example, below is a snapshot of how your application could look like when the user clicked on three different places on the application.

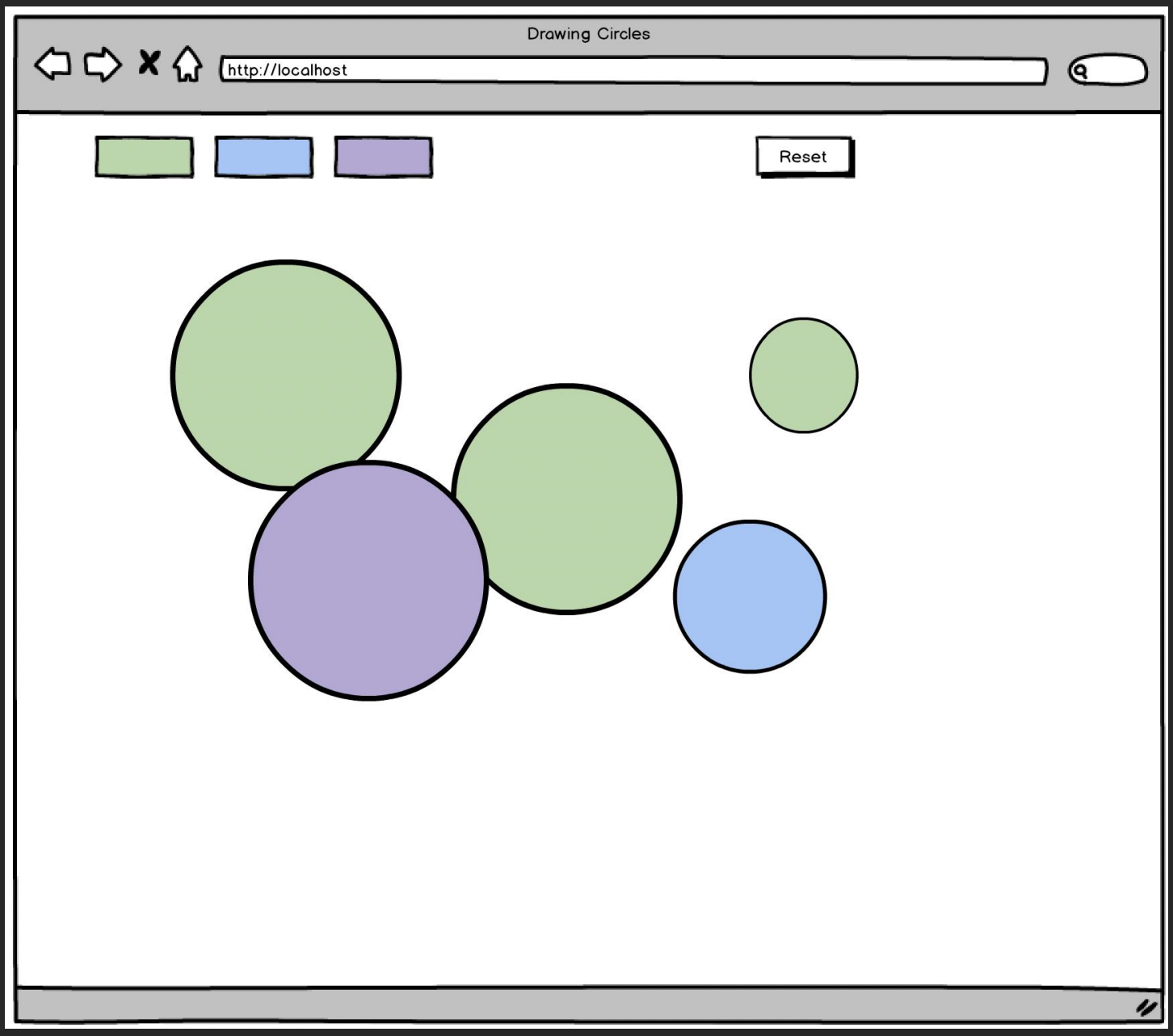


There are various ways to draw on the HTML. When you're drawing vectors, you can use SVG or canvas. These are beyond the scope of this chapter, so do NOT use SVG or canvas. At a high level, SVG can be used to draw vector graphics. Canvas can be used to display 2D images. For now, these circles could all be just plain old HTML with fixed width and height and with border-radius. For example,

```
<p style="position:absolute; top:150px; left:75px; width:200px; height:200px; border-radius:10
0px; background-color: #CCE8CC"></p>
<p style="position:absolute; top:230px; left:275px; width:150px; height:150px; border-radius:7
0px; background-color: #CCE8CC"></p>
<p style="position:absolute; top:100px; left:475px; width:180px; height:180px; border-radius:9
0px; background-color: #CCE8CC"></p><br>
```

Once you get this to work, let's evolve the application to have additional event listeners.

Now, have the default color still be green but display three color choices on the top left. When the appropriate color is chosen, have the chosen color be highlighted (either by putting a thicker border or by slightly changing how the chosen color box looks). Then when the user clicks somewhere on the screen, the newly created circle should be what was chosen by the user. Also have a 'Reset' button that resets the screen, allowing the user to start from scratch.



The goal of this assignment is to get you to be more familiar with Event Listeners using pure Javascript. Therefore, do NOT use jQuery for any of this assignment. Use plain Javascript to do this now.

Extra Credit (required for V88 trainees)

Now, update your code so that the created circles slowly shrink and once it reaches a radius of 0, have it disappear completely (not just on the screen but also on the HTML DOM).

Remember that when you learned jQuery, you used callbacks all the time! For example, do you remember doing things like this?

```
$(document).click(function() {
  ...
});
```

Or how about any of the following?

```
$('h2, h3').click(function() {
   $('h2, h3').hide();
}
$.get("/...", function(res) {
...
});
```

Note that in all of these, you were passing a function as an argument to another function. You were using callbacks!

Now, instead of just using these functions where you were familiar with passing a callback function, it's your turn to create these functions yourself! This time, instead of using jQuery, we want you to use native Javascript to build these functions.

JS dQuery

Now that you know how to use a callback, let's have you create a javascript library and name it dQuery.js. Anyone who loads your script could would have $query available where they could execute codes such as below. Note that what we want you to create is essentially a mini library, similar to jQuery, but instead where you wouldn't use any jQuery but build all of this from scratch using native Javascript. Again, please do NOT use jQuery to build this assignment but really see how you could create a library like jQuery, but from scratch.

```
<html>
<head>
<title>JS Dollar Query</title>
   <script src="dQuery.js"></script>
</head>
<body>
   <h1>Heading 1</h1>
   <p>Sample Paragraph 1</p>
   <p>Sample Paragraph 2</p>
   <button id='show_all'>Show All</button>
   <button id='hide_all'>Hide All</button>
   <script>
      //clicking on h1 should hide all the h1
      $query('h1').click(function() {
          console.log('h1 is clicked');
          $query('h1').hide();
```

```
    });

    //clicking on p should hide all the paragraphs
    $query('p').click(function() {
        console.log('p is clicked');
        $query('p').hide();
    });
    /clicking on #show_all should show both h1 and p
    $query('#show_all').click(function() {
        console.log('#show_all is clicked');
        $query('h1').show();
        $query('p').show();
    });
    //clicking on #hide_all should hide both h1 and p
    $query('#hide_all').click(function() {
        console.log('#hide_all is clicked');
        $query('h1').hide();
        $query('p').hide();
    });
  </script>
</body>
</html>
```

This is if they were using your library for the front-end (for the browser to execute). You could also have your library used for the back-end and how to do this will be taught later.

To help you get familiar with both ES5 and ES6, create three different versions for this javascript file.

- First one - where you're using ES5 but not using prototype
- Second one - where you're using ES5 and prototype
- Third one - where you're using ES6

You may need to do a bit of Googling to learn how to show/hide a html element (but it should take you no more than a few minutes of research to learn how to do this using native Javascript).

Advanced Challenge (required for all Village88 trainees)

Whenever an element is clicked, the browser sends information about the click event. Make this information also available to your callback function. For example,

```
$query('#hide_all').click(function(event) {
  console.log('event passed to the callback function is', event);
}
```

Really think about what's happening here and who is sending information back to the callback function. If you understand what's happening here, you are quite close to creating a framework like jQuery yourself too! This is good as good developers focus more on the fundamentals and how things work, rather than spending time learning about how other people wrote things and how to use other people's codes. Best developers can create anything, even frameworks like jQuery, React, Angular, etc. Mediocre developers (or developers who haven't really challenged themselves yet) sometimes get too complacent and focus too much on studying other people's codes or libraries (e.g. React, Angular,

jQuery, etc) instead of learning what's happening behind the scene and how they too can create these frameworks themselves.

Challenge yourself always to really understand what's happening behind the scene and it will make you a better developer. Note that all of these callback features that you were using with jQuery and other javascript library were all due to the simple fact that in Javascript you could store a function in a variable! This simple fact brings out some interesting nature that you'll learn more as you progress through the course.